ValidateThis 0.94 - More community contributions

Posted At : June 5, 2010 7:58 AM | Posted By : Bob Silverberg
Related Categories: ValidateThis

I've just released version 0.94 of ValidateThis, my validation framework for ColdFusion objects. Once again this update includes community contributions, including some from **Jamie Krug** and **John Whish**. Here's a summary of the changes, followed by the details for each one.

- A new *boolean* validation type has been added.
- Proper optionality is now supported for all validation types.
- The framework can locate your rules definition file with zero configuration when you pass an object into a method call.
- A *newResult()* method has been added to the ColdBox plugin.
- A *getFailureMessages()* method has been added to the Result object.
- An issue with overriding failure messages with the *custom* validation type has been addressed.

The latest version can be downloaded from **the ValidateThis RIAForge site**. Details of the enhancements follow:

New Boolean Validation Type

Jamie Krug had a need for a boolean validation type in a project, so he went ahead and created one and contributed it back to the framework. Originally only the server-side piece was implemented, so I solicited some help from the ValidateThis community to come up with a JavaScript function that could check whether a value is a valid ColdFusion boolean. I received some feedback on this from Adam Drew, TJ Downes and John Whish. John's regex solution ended up as the clear winner. I've included the code below in case anyone is interested, including a script that demonstrates the output:

```
<script type="text/javascript">

function isBoolean( value )

{

 if ( value==null )

 {

  return false

 }

 else

 {

  var tocheck = value.toString();

  var pattern = /^((-){0,1}[0-9]{1,}(\.([0-9]{1,})){0,1}|true|false|yes|no)$/gi;

  return tocheck.match( pattern ) == null ? false : true;

 }

}


document.write('these can all be treated as boolean in CF<br />');


document.write(isBoolean('YES'));

document.write(isBoolean('True'));

document.write(isBoolean(true));

document.write(isBoolean('1'));

document.write(isBoolean(1));

document.write(isBoolean(2));

document.write(isBoolean('2'));

document.write(isBoolean('NO'));

document.write(isBoolean('false'));

document.write(isBoolean(false));

document.write(isBoolean('0'));

document.write(isBoolean(0));


document.write('<br />these can NOT be treated as boolean in CF<br />');


document.write(isBoolean({}));

document.write(isBoolean(null));

document.write(isBoolean('Foo'));

document.write(isBoolean(' TRUE '));

document.write(isBoolean(10));

document.write(isBoolean('10'));

document.write(isBoolean(' NO '));

document.write(isBoolean(' false '));

document.write(isBoolean(''));

document.write(isBoolean('{ts 2010-02-01}'));

</script>
```

Proper and Consistent Optionality

In previous releases certain validation types were optional, meaning that the validation wouldn't fail if the property was left blank. This option was implemented sporadically, so I wanted to address that. Now all validation types are optional by default, meaning that empty properties will not trigger a validation failure. However, the framework is smart enough to know that if a property is also required (i.e., it has a rule defined on it with a type of *required*), then the validation in question will be treated as mandatory. Perhaps that's a bit confusing, so let's look at an example.

Imagine an object with an *emailAddress* property, and the following rules defined:

```
<objectProperties>

 <property name="emailAddress">

  <rule type="email" />

 </property>

</objectProperties>
```

If I were to place the value "bob" into the emailAddress property and then validate the validation would fail as "bob" is not a valid email address. I would get a single validation failure saying "The Email Address must be a valid Email address." If, on the other hand, I were to store an empty string in the emailAddress property, I would get no validation failures as the validation is treated as optional.

Let's change the rules for emailAddress to look like this:

```
<objectProperties>

 <property name="emailAddress">

  <rule type="required" />

  <rule type="email" />

 </property>

</objectProperties>
```

I would get the identical single failure message if I were to place the value "bob" into the emailAddress property, but if I were to store an empty string in the emailAddress property, I would now get two validation failure messages:

1. The Email Address is required.
2. The Email Address must be a valid Email address.

To sum up, a validation rule will be treated as optional, unless the property in question is required.

Zero Configuration Locating of Rules Definition Files

In previous releases of the framework you told VT where to find your rules definition files using the *definitionPath* key of the ValidateThis Config struct. Over the course of developing the framework this ability has been enhanced to support multiple paths, and there has always been a built-in default of */model*. John Whish was interested in removing the requirement to configure this location, particularly when the files may exist in multiple different locations. The demo for VT that uses Transfer and integrates VT directly into the business objects is already designed to do this - it assumes that your rules definition files will be in the same folder as your object cfcs, and it simply picks them up from there. That logic is coded into the demo, rather than being available in the framework, so I added a few lines of code to support that.

Now, when you call a method on the framework into which you pass in the object to be validated (e.g., the *validate()* method), the framework will first attempt to locate a rules definition file in the same folder as the cfc that defines the object. If it doesn't find one there, then it will search through the definitionPaths to find one. So, if you choose to store your rules definition files alongside your cfcs, VT will be able to automatically locate those files without requiring you to specify their location. For more information on how VT uses the definitionPath setting, take a look at theValidateThis Online Documentation.

New newResult() Method in the ColdBox Plugin

Because the ColdBox plugin makes use of onMissingMethod, it was already possible to call *newResult()* on the plugin to get a new, empty Result object. In order to make the API for the plugin more obvious, I had added a number of methods to it that could have simply been handled by onMissingMethod. To keep the API up to date I have now added an explicit newResult() method.

New getFailureMessages() Method in the Result object

Another suggestion from John Whish prompted me to add a *getFailureMessages()* method to the Result object. This method will return all validation failure messages as an array of strings. I believe that John plans to use this in conjunction with ColdBox's MessageBox object to enhance the ColdBox plugin to return a configured MessageBox.

Overriding Failure Messages with the Custom Validation Type

I'm guessing that this one will be meaningless to pretty much everyone, but I'm a stickler for documenting changes. I'll just sum it up by saying that now, if you have a *custom*validation rule, and you've defined an explicit *failureMessage* for the rule, the framework will override that message with any message that is generated by the method called for the custom validation. It was not doing that in the past, and I deemed that a bug.

Once again, the latest code is available from the ValidateThis RIAForge site, and if you have any questions about the framework, or suggestions for enhancements, please send them to theValidateThis Google Group. I'd also like to thank Jamie and John once again for their contributions to the framework.